Designing a data streaming infrastructure for a smart city crowdsensing platform

Aleksa Miletić

Faculty of Organisational Sciences
University of Belgrade
Belgrade, Serbia
aleksa.miletic@elab.rs
0000-0001-8940-9897

Petar Lukovac

Faculty of Organisational Sciences
University of Belgrade
Belgrade, Serbia
petar.lukovac@elab.rs
0000-0003-4561-8886

Branislav Jovanić

Intitute of Physics
University of Belgrade
Belgrade, Serbia
brana@elab.rs
0000-0003-4130-1638

Božidar Radenković

Faculty of Organisational Sciences
University of Belgrade
Belgrade, Serbia
boza@elab.rs
0000-0003-2111-7788

Abstract— This article considers the problem of designing data streaming infrastructures for crowdsensing systems. The goal is to propose an infrastructure that would provide a scalable and seamless collection of various data in smart city, such as noise, vibrations, air quality, citizens' health parameters, etc., streaming collected data into the cloud infrastructure, and preparing it for further analysis. The paper focuses on the aspects of modelling data streams, and the implementation using the Apache Kafka software. The prototype of the proposed system is implemented within the private cloud infrastructure of the Department of e-business, Faculty of Organizational Sciences.

Keywords - data streaming, crowdsensing, smart city, IoT

I. INTRODUCTION

Streaming data can be beneficial to any industry that deals with big data. It is used to integrate, process, filter, analyze and react to collected data as it is received in real-time. Today, it is used in many applications such as online-shopping, recommendations on streaming services, fraud detection etc.

Data stream refers to an ordered sequence of data, which is unbounded and continuously updated. It typically stands for a continuous transfer of a large quantity of small data, from a data producer to a data consumer. Data in data streams are processed sequentially and incrementally, one at a time or in small batches. Processed data streams are then used for further analysis, using various algorithms [1].

Data streams can be found in various business activities and domains, including banking, commerce, production, healthcare etc. A specific importance was put on data stremaing architectures with the development of Internet of things, since sensors typically generate numerous streams of data, which are then transfered to a cloud storage [9].

Comparing to traditional batch processing, data streaming approaches provide data in near real time, allowing for real-time analytics and quicker insights. Using streams, data can be frequently updated, and the users can be notified about the changes of interest. Unlike the traditional request-response paradigm, it focuses on push notifications to users, without explicits users' requests.

The aim of this paper is to point out the problems that exist when designing data streams, to give certain guidelines on how to solve these problems, and to define system architecture for measuring noise, vibrations in traffic, allergens, and air quality in smart cities.

II. THEORETICAL BACKGROUND

Data stream is an abstraction representing an infinite and ever-growing dataset. The dataset is infinite and ever-growing because over time, new records keep arriving [4].

Event streams are arranged. The events have an order in which they are performed, but it is difficult to anticipate which event will perform next. On the example of crowdsensing system about the amount of allergens in the air, the easiest scenario is to show harmfulness of air in a certain part of the city, but there is possibility for the user to enter their chronic and seasonal diseases and to be notified if they are near, for them, harmful part of the city. The difference between event streams and database tables is that the database tables are considered unordered, and the „order by" clause is not part of the relational model, but had been added to make data easier to display [4].

Immutable data records. Once events are triggered, they can't be changed anymore. If the user updates their chronic and seasonal diseases, the old ones won't be deleted. The difference between database tables and event streams is that we can update or delete data in the database tables, but they are considered as an additional transactions. Instead, we can store transactions in a data stream [4].

Event streams are replayable. This is a good feature of event streams because there would be a problem if there are events which occurred a few months earlier, or if we manage to change it, there would be no record that that event ever happened before [4].

The data that are sent through the stream are different from system to system. The size of events that are sent through stream can be very small and can be huge. Data that are sent can be key-value pairs, JSON structures etc. In Kafka there are topics which are used to organize messages and each topic has unique name. There are producers which send data to the topic, and then they are redirected to subscribed consumers. Producer is an application that represents the source of the data stream. Consumers are applications that consume records from Kafka cluster.

III. DATA STREAM MODELLING

Apache Kafka is a data streaming platform whose ability is to process several trillion requests in one day [6]. Today, Kafka is used for streaming data in real-time, storing a large amount of data, and analyzing the same data. The advantage of the Apache Kafka tool is the ability to store data on Kafka, which can actively monitor the transaction execution process. Unlike the MQTT or AMQ protocol, Apache Kafka allows users to receive messages that are important to them. In this case, a user will receive data only when he wants to list them.

Topics are like folders in a filesystem, and events are files in that folder. Kafka is a multi-producer and multi-subscriber, so it can receive data from zero or more producers and send it to zero or more consumers. Data from a topic can be read whenever there is a need for it. Also, the data is not deleted immediately after use, but a certain period is defined in which data is stored, and after that period it is removed. Kafka's performance is effectively constant with respect to data size, so storing data for a long time is perfectly fine [7].

The main question in topic design is: How do we decide on which topic to send the next event? If topics aren't designed in an appropriate way it can later bring us to redesign topics, that is creating a topic model from the beginning. Bellow will be described good practices for creating topics.

As mentioned, Kafka is a platform for streaming data, and one of the main things which need to be designed in Kafka are topics. Topics are an ordered collection of events that are stored in a durable way, which means they are written to disk and they are replicated. There are problems in topic design: whether different types of data should be put into one topic, whether a couple of topics can be merged if they are logically connected, how to protect sensitive data sent to a topic, etc. The answers to these and some similar questions will be given below.

According to Martin Grotzke and Daniel Orner, Kafka topics should be designed as follows [2][3]:Cases for which topics should be designed:

- Topics can be separated by domains and subdomains to which events relate. If two events are logically different then they should be put in different topics. When a topic is created, there should also be created object schema, and only objects which satisfy that schema could go through the topic. Each topic should represent one type to work with. Meaning is lost if there are mixed types in one topic.
- If there is a strong connection between two events, should consider whether they can go to the same topic. Also, if there is an exact order in which events should be called, then they should go in the same topic.
- If we withdraw some data, among which there is sensitive data of some user, that data should be separated from the others, and therefore a new topic should be created in which they would be placed.
- Frequently triggered events (e.g. a sensor that sends data multiple times per second) should be placed in special topics [2]. In most cases, you'll want to have exactly one producer and one or more consumers. If more services need to send the same kind of data, that's a clue that your service boundaries are not well-drawn, or that one service should be talking to the other one in some different way instead. This applies to event, entity, and response topics.
- In some cases, you should have the opposite: one or more producers and exactly one consumer. This applies to request topics, ensuring only one system processes each request. You can think of it as a funnel instead of a branch-out.
- Standardize topic names to make it simple when creating new ones. One possible format is {BoundedContext}.{Subdomain}. For example: Noise.NoiseCreatedEvent, Allergens.ParticleSize, AirQuality.Temperature.

Maximal number of topics is infinite. Minimal number of topics should be near number of different things which are send from producer to consumer.

Table 1, relationship between few and a lot of topics[5]

Less topics	***More topics***
Consumer may have to filter messages	Consumers can read only topics they care about
Less processing overhead of managining masters and consumers	Slower restarts, other processing overheads
Less configuration to manage	More flexibility in configuration

Streaming is becoming more and more popular because it is not necessary to store a large amount of data, but it can be quickly analyzed and gain insight into the data. The most important thing is to determine the current data, whether it is 2 minutes, 1 day, or maybe a certain number of events (100). The time period we need is called windowing.

There are methods which define length of window [10]:

- Tumbling window - these are windows that have a clear duration and are constant. This means that after the completion of one, the next one continues.
- Sliding window - these are windows that are timed, but it is possible to overlap multiple windows.
- Session window - does not depend on the period, but on the activity of the event in a particular period. This means that certain windows have a shorter period if the time flow is longer.
- Events can be selected by occurrence time:
- Event-time represents the time of occurrence of the event.
- Processing-time represents the time when the event was processed by the system

There must be a certain period of time that would indicate that certain data is current and that is useful for the system. If a certain event is delayed when arriving in the system by more than the defined value, it will not be included in the set of events for processing. This time period is called the watermark [10].

IV. DATA STREMING INFRASTRUCTURE FOR THE SMART CITY CROWDSENSING SYSTEM

The goal of the system proposed in this paper is to provide an adequate data streaming infrastructure that would serve for crowdsensing data in the smart city (Figure 1). The proposed system can support a large number of IoT subsystems as producers, as well as a large number of various consumer applications. Within this paper, only a few examples of IoT subsystems are presented: measuring noise, measuring vibrations, measuring air quality, and measuring allergens.

Within the proposed system, measurements are done using crowdsensing, where a large number of participants collect and provide data. Collected data is sent to the cloud, and accepted using an ingress service. Collected data streams are then transferred to the back end of the system, where they are being transformed and prepared for further analysis. Processed and analyzed data are delivered to consumers through various delivery models (pull, push) and through various applications. Client applications are connected to the system through API of access services.

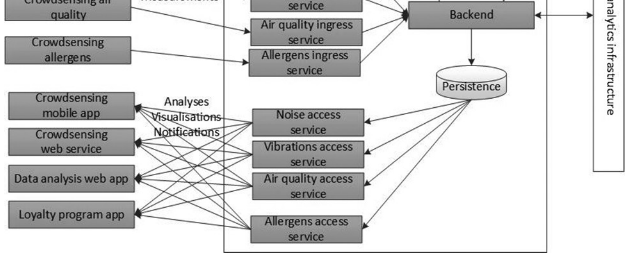

Figure 1 – The overview of the proposed system

The main components of the proposed system are [8]:

- IoT devices are used for collecting data from sensors. If needed and possible, data measured from each sensor is processed and transformed at the edge device, and then sent to the cloud.
- Mobile applications for crowdsensing are used to collect measurements using participants' mobile devices. They provide collecting data such as noise at micro-locations, vibrations in public transportation and traffic, etc.
- Data streaming infrastructure includes several components: 1) ingress services for every device with different measurements, i.e. a service that receives data from IoT systems and mobile applications; 2) stream processing cluster, which enables the transformation of data streams; 3) access services for every type of measurement which enables communication with consumers of the data, i.e. applications that use the data streams.
- Data analytics infrastructure provides services for real-time analytics. It enables the implementation of advanced analyses, machine learning techniques, or other predictive models.
- Consumer applications include various mobile, smart watch, or other applications that consume crowdsensed data in any form. They usually present data using adequate visualizations and enable notifications.

V. IMPLEMENTATION DETAILS

Table 2 represents Kafka topics used for implementing the proposed system. Within each topic, data is streamed in a form of key-value pairs.

Table 2. Specification of topics within the proposed system

Topic	***Description***
raw_noise_ measurements	The topic for accepting traffic measurement data. The following data will be stored: devices that measure noise volume (sensors - name and type of sensor and mobile phone - mac address) - latitude and longitude, user from whose device noise is measured - username, name, surname, measured frequency - volume frequencies, municipality - the name of the municipality
raw_noise_ measurements _anonymized	Anonymous data when measuring traffic noise
average_noise_ time_location	Calculations of averages for a location at selected time periods; a topic is created for each combination of predefined time slots (daily, hourly) and selected locations of interest.
max_noise_ time_location	Maximal measured values for selected time periods at a location.

raw_vibration_ measurements	The topic for accepting data from traffic vibration measurements. The following data will be stored: devices that measure vibrations (sensors - name and type of sensor and mobile phone - mac address) - latitude and longitude, a user from whose device the vibration is measured - username, name, surname, measured frequency - frequency, municipality - the name of the municipality
raw_vibration_ measurements _anonymized	Anonymous data when measuring vibrations in traffic.
average_vibration_time_location	Calculations of averages for a location at selected time periods; a topic is created for each combination of predefined time slots (daily, hourly) and selected locations of interest.
max_vibration_time_location	Maximal measured values for selected time periods at a location.
raw_airQuality_measurements	The topic for accepting air quality measurements. The following data will be stored: devices that measure air quality (sensors - name and type sensor) - latitude and longitude, air quality measured for a short period of time, municipality - the name of the municipality
raw_airQuality_measurements_anonymized	Anonymous data when measuring air quality.
average_ airQuality_ time_location	Calculations of averages for a location at selected time periods; a topic is created for each combination of predefined time slots (daily, hourly) and selected locations of interest.
max_airQuality_time_location	Maximal measured values for selected time periods at a location.
raw_allergens_ measurements	The topic for accepting allergen measurements. The following data will be kept: place where measured - name, address, latitude and longitude, weather forecast for that place - temperature, humidity, rain level, and cloud level, wind type - name, direction, and strength of wind blowing, device for measuring allergens - name, type of particle measured by the device - name, a measurement performed by the device - measured value.
raw_allergens_ measurements_ anonymized	Anonymous allergen measurement data.
average_allergens_time_location	Calculations of averages for a location at selected time periods; a topic is created for each combination of predefined time slots (daily, hourly) and selected locations of interest.
max_allergens_time_location	Maximal measured values for selected time periods at a location.

VI. Conclusion

In this paper, the focus is on defining the architecture when creating Kafka topics. The article indicates the most common mistakes and the most common problems that developers encounter when creating Kafka topics and is pointed out how to prevent these problems. It also explains what data streaming is and the focus is on Apache Kafka which is one of the most used tools for data streaming, as well as the key concepts of Apache Kafka: consumer, producer, watermarking, and windowing.

The plan of future development is the development of an IoT crowdsensing system based on data streaming architecture, using Apache Kafka, and based on the model presented in this paper. Sensor data will also be collected and sent via the MQTT protocol to Apache Kafka, where it will be processed and displayed to users in mobile applications.

References

[1] Apache Kafka (2017) [Online] https://kafka.apache.org/10/documentation/streams/core-concepts.html

[2] M. Grotzke, “Kafka Topic Design Guidelines”, https://inoio.de/blog/2021/05/21/kafka-topic-design-guidelines/, 21. May 2021

[3] D. Orner “Schema and Topic Design in Event-Driven Systems (featuring Kafka!)”, https://medium.com/flippengineering/schema-and-topic-design-in-event-driven-systems-featuring-kafka-a555ddfdb8d8, 1. May 2020

[4] N. Narkhede, G. Shapira, T. Palino “Kafka: The Definitive Guide: Real-Time Data and Stream Processing at Scale”, 1st ed. O’Reilly Sebastopol, pp. 248-250, July 2017.

[5] B. Slater “Topic Design” (August 2018) [Webinar]. Instaclustr. https://www.youtube.com/watch?v=XQJ9NoX_yyU&ab_channel=Instaclustr

[6] Н. Кудуз, О. Ћалић, and С. Поповић, Моделовање система за управљање токовима порука примјеном Apache Kafka, 2019.

[7] Apache Kafka (2017) [Online] https://kafka.apache.org/documentation/#gettingStarted

[8] A. Labus, M. Radenković, S. Nešković, S. Popović and S. Mitrović (2022) “A Smart City IoT Crowdsensing System Based on Data Streaming Architecture” Smart Innovation, Systems and Technologies, vol 279. Springer, Singapore.

[9] R. Kraft et al. “Efficient Processing of Geospatial mHealth Data Using a Scalable Crowdsensing Platform” Sensors 2020

[10] J. Traub et al., "Scotty: Efficient Window Aggregation for Out-of-Order Stream Processing," 2018 IEEE 34th International Conference on Data Engineering (ICDE), 2018, pp. 1300-1303, doi: 10.1109/ICDE.2018.00135.